# MergeCOM-3™ C/C++ DICOM Toolkit

## Message Database Manual

## V4.4.0

**Copyright 2011 Merge Healthcare Incorporated**
**Unauthorized use, reproduction, or disclosure is prohibited.**

This document has been prepared by Merge Healthcare Incorporated, for its customers. The content of this document is confidential. It may be reproduced only with written permission from Merge Healthcare. Specifications contained herein are subject to change, and these changes will be reported in subsequent revisions or editions.

Merge Healthcare® is a registered trademarks of Merge Healthcare Incorporated.

*MergeCOM-3*™ is a trademark of Merge Healthcare. The names of other products mentioned in this document may be the trademarks or registered trademarks of their respective companies.

Merge Healthcare
6303 Airport Road, Suite 500, Mississauga, ON, Canada L4V 1R8
Tel: +1-905-364-8000 • Fax: +1-905-364-8100
Toll-free in North America: 1-800-724-5970

For assistance, please contact Merge Healthcare Customer Support:

- In North America, call toll free 1-800-668-7990, then select option 2
- International, call Merge Healthcare (in Canada) +1-905-672-7990, then select option 2
- Email mc3support@merge.com

COM-56 Rev 1.0

# Contents

CONTENTS III

OVERVIEW 1

Message Dictionary Components ..... 1

Conventions ..... 4

DICTIONARY AND MESSAGE PROFILES 5

Using mc3dcomb ..... 6

Using mc3icomb ..... 7
Format of mc3icomb's configuration file ..... 7
Final overview of the process ..... 10

GENERATING THE BINARY DATABASE 10

Dictionary File Generation (mc3dict) ..... 10

Message Info File Generation (mc3info) ..... 11

ACCESSING PRIVATE ATTRIBUTES 12

APPENDIX - DICTIONARY PROFILE SPECIFICATIONS 14

Data File Descriptions ..... 14

Private Attribute Specification ..... 15

Standard Attribute Specification ..... 16

Private Attribute Specification ..... 17

Example message info profile entry ..... 18

Extending the Message Info Profile ..... 19

# Overview

This manual is targeted toward a special class of advanced developers of DICOM applications using the MergeCOM-3 DICOM Toolkit that wish to:

- supplement the standard MergeCOM-3 message dictionary with private extensions; and
- verify these private extensions through the message validation functionality provided by MergeCOM-3 DICOM Toolkit.

MergeCOM-3 DICOM Toolkit supports the addition of private attributes at runtime through the use of the `MC_Add_Private_Block()`, and `MC_Add_Private_Attribute()` calls. This method is sufficient for most toolkit users where private attributes are encoded or accessed in a limited manner.

But, for more advanced user's (e.g. OEM customers) that maintain large groups of private attributes across product lines, it is desirable to maintain these private attributes in the MergeCOM-3 DICOM Toolkit Dictionary Files. Advantages of this include:

- ability to maintain Private Dictionary extensions on-site, under appropriate source-code control.
- ability to create product-specific message dictionaries.
- avoiding the overhead of `MC_Add_Private_...()` calls before using private attributes.
- More 'built-in' knowledge about your own private attributes that are streamed in from DICOM messages over the network or from media.
- Ability to validate your message objects against private extensions using the existing `MC_Validate_Message( )` call.

**Impacts on DICOM Conformance**

When adding Private Attributes to a standard DICOM message object (SOP Class) you are creating a Standard Extended, Specialized, or Private SOP Class, depending on the rules you follow. Part 2 of the Standard details these rules and how each of these types of SOP classes effect conformance. Part 2 also explains how to document these extensions within your DICOM Conformance Statement. Be sure to read and understand Part 2 of DICOM before adding Private Attributes to DICOM messages.

If the terminology being used here is confusing you need to read and understand the other MergeCOM-3 DICOM Toolkit User Manual, and refer to the DICOM Standard itself, before continuing on with this manual. See the Documentation Roadmap section below.

## Message Dictionary Components

When the MergeCOM-3 DICOM Toolkit validates a message through the `MC_Validate_Message( )` call it accesses several binary dictionary files. These binary files are accessed during runtime via the file system. One of these files, the DICOM Data Dictionary file, can also be kept in memory or statically linked into your application as an object module. See Figure 1.

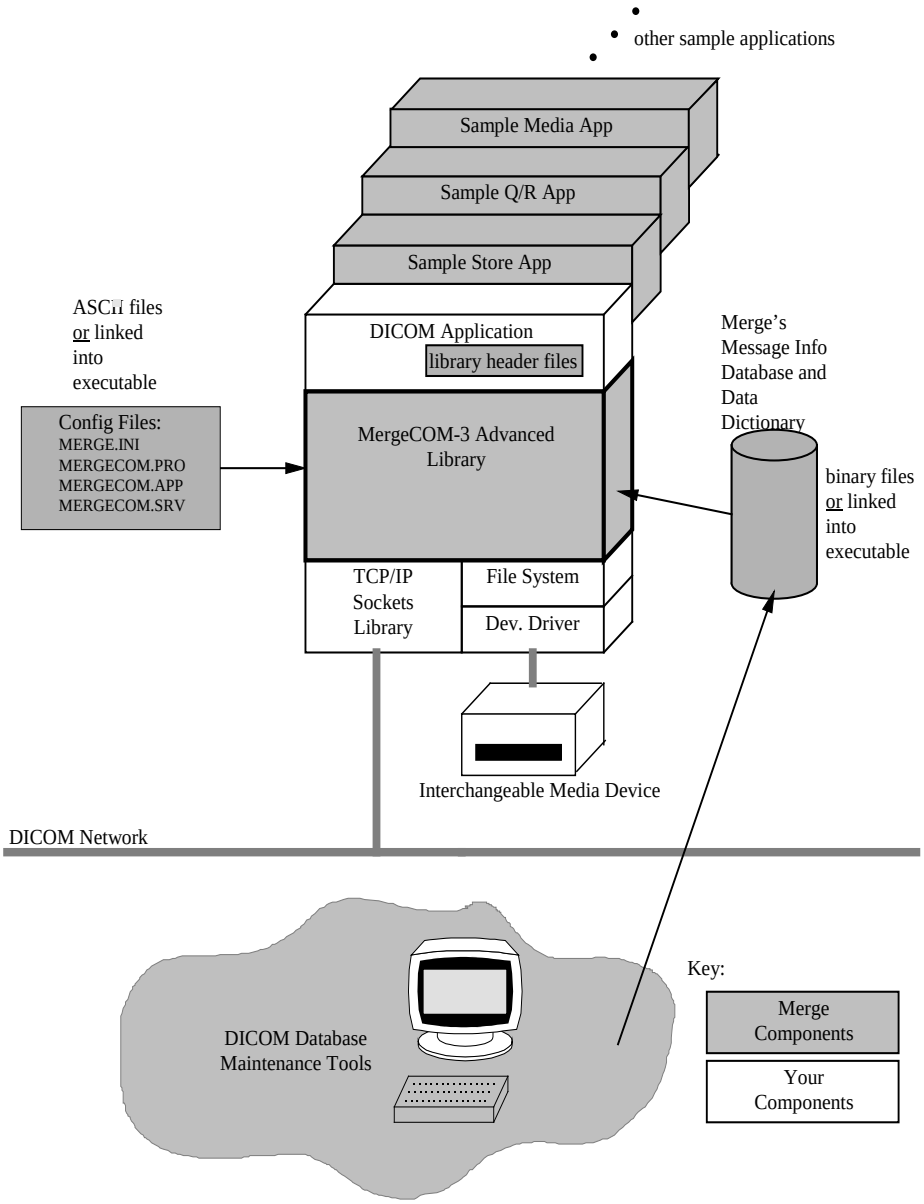


*Figure 1: MergeCOM-3 DICOM Toolkit with Merge Healthcare Components Highlighted*

For most applications, accessing the binary files is preferred because this requires less static memory and is easier to reconfigure (does not require rebuilding executables). But, in embedded systems, where a file system is not available, statically linking in the Data Dictionary file allows attribute level (but not message level) validation, without requiring a file system.

The MergeCOM-3 DICOM Toolkit configuration files can also be statically linked into to your executable. Your application specifies how it wishes to access the dictionary and configuration files using the `MC_Library_Initialization( )` call. See the Reference Manual for further details.

Merge may add the ability to statically link in message info files in the future, but at present the memory cost associated with this approach makes its appeal limited. It is most often the case, at least in embedded environments, that

development and test are done with a file system (allowing for message info files) and message-level validation is not done in the final production environment.

Normally, the library uses the binary files and object files shipped with the toolkit. It is also possible for advanced developers to extend or specify their own private data dictionary and message info files. This document outlines the structure of the toolkit dictionary and message info files and describes the use of utilities needed to generate new binary dictionary files.

The MergeCOM-3 DICOM Toolkit documentation is structured as pictured in Figure 2.

**Read Me FIRST!**

The User's Manual is the foundation for all other documentation because it explains the concepts of DICOM and the DICOM Toolkit. Before plunging into the Reference Manual or Sample Application Guides, you should be comfortable with the material in the User Manual.

The Reference Manual is where you go for detailed information on the DICOM Toolkit. This includes the Application Programming Interface (API), toolkit configuration, the runtime object database, and status logging. The Reference Manual also includes a DICOM conformance statement for the toolkit.

The DICOM Message Database Manual is an optional extension that describes the organization of the MergeCOM-3 DICOM Database and how to use it to extend standard services and define your own private services. Tools are supplied for converting the contents of the database into the binary runtime object database.

**Sample Applications**

The Sample Application Guides describe approaches to developing specific classes of DICOM applications (Image Transfer, Query/Retrieve, Print, HIS/RIS, Storage Media, etc.). They highlight pertinent information from Parts 3 or 4 of the DICOM Standard in a more readable way and in the context of the DICOM Toolkit. The Application Guides also detail the DICOM messages that can be passed between applications on the network. Also, a sample application is described and the application supplied in source form for your platform.

Platform-specific information required to use the DICOM Toolkit on your target platform are specified in Platform Notes. This includes supported compilers, compiler options, link options, configuration, and run-time related issues.

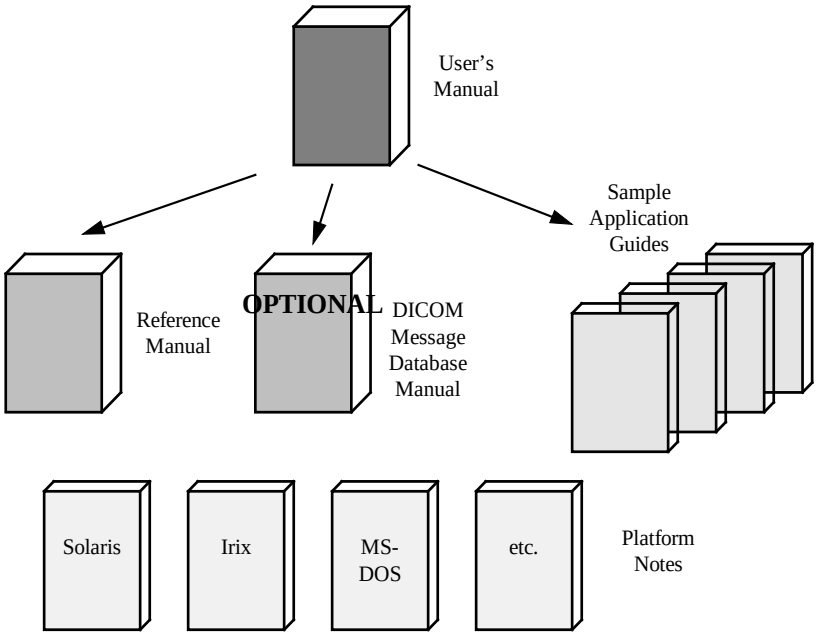


*Figure 2: MergeCOM-3 DICOM Toolkit Documentation Roadmap*

## Conventions

This manual follows a few formatting conventions.

Terms that are being defined are presented in **boldface**.

**Sample Margin Note**

Margin notes (in the left margin) are used to highlight important points or sections of the document.

Sample commands appear in **`bold courier`** font, while sample output, source code, and function calls appear in `standard courier` font.

Hexadecimal numbers are written with a trailing H. For example 16 decimal is equivalent to 10H hexadecimal.

***Performance Tuning***

Portions of the document that can relate directly to the performance of your application are marked with the special margin note ***Performance Tuning*.**

# Dictionary and Message Profiles

In Figure 1 we detailed the operation of the MergeCOM-3 DICOM Toolkit and how it references binary dictionary and message info files or objects to perform runtime message validation.

### Creating The Dictionary File

The binary data dictionary file is named `mrgcom3.dct` and contains a binary coded representation of the DICOM data dictionary (as specified in Part 6 of the DICOM Standard). This includes all the standard attributes, their tags, names, value multiplicities (VM's), and value representations (VR's). A utility called `mc3dict` is used to create this binary dictionary file on a specific platform from an ASCII input file (usually named `diction.pfl`). The ASCII input profile is platform independent and developers can modify `diction.pfl` or create their own dictionary profile and run them through `mc3dict` to generate a customized `mrgcom3.dct`.

### Creating The Message Info Files

The binary message info file has the name `mrgcom3.msg` and contains a binary coded representation of DICOM Service Object Pairs (SOP's. These SOP's specify the attributes (contained in the data dictionary) that make up a message or item, their value type (VT), and any enumerated values or defined terms. Items that are used in attributes of VR Sequence of Item (SQ) are also specified in message info file. A utility called `mc3info` is used to create this binary message info file on a specific platform from a single ASCII input file (usually named `info.pfl`).

Figure 3 represents this conversion process for both the data dictionary and message profiles.

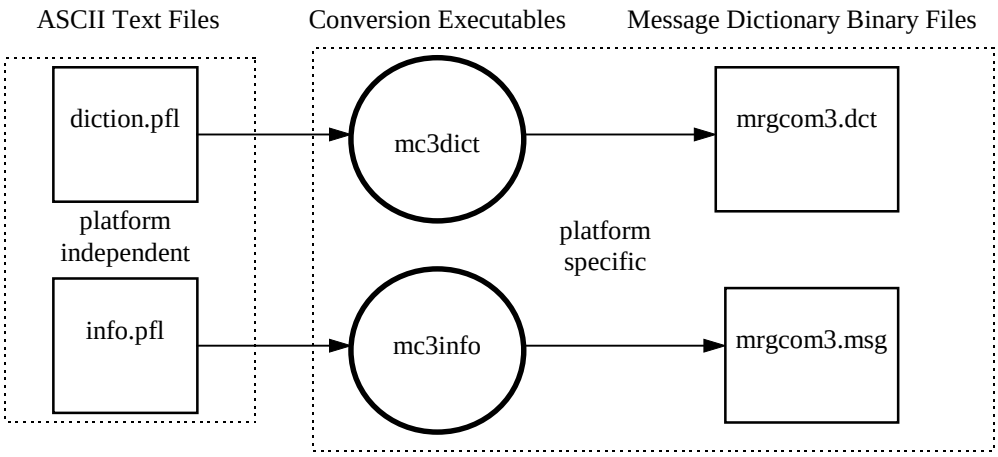


*Figure 3: ASCII to Binary Message Dictionary Conversion Process*

MergeCOM-3 DICOM Toolkit also supports a `gendict` utility program that converts the binary dictionary file into a source code module that can be compiled and linked into your application. Documentation of this conversion process can be found in the Toolkit Reference Manual.

The remainder of this document describes two utility programs `mc3dcomb` and `mc3icomb`. These two utility programs, along with the existing `mergecom.srv`, `diction.pfl` and `info.pfl` files are used to perform custom modifications to the dictionary and message databases.

It is important to understand the layout of the "`*.pfl`" files before attempting to use the two combine utilities. This document also provides an appendix which describes the manual process of extending the database by directly manipulating the `mergecom.srv`, `info.pfl`, and `diction.pfl` files.

While Merge recommends that the combine utilities be used to perform dictionary changes rather than manually changing the `mergecom.srv`, `info.pfl`, and `diction.pfl`, it may be helpful to understand the layout of these files and how the toolkit makes use of them.

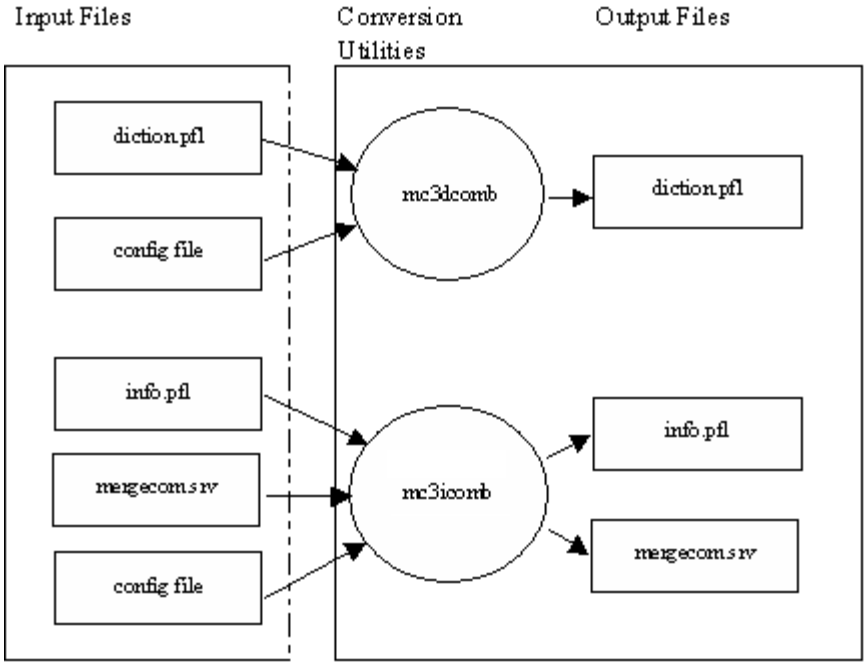


*Figure 4: Input files and output files for the dictionary utility programs.*

## Using mc3dcomb

The "`mc3dcomb`" utility is used to merge the contents of two `diction.pfl` files together. One of the two files could be the included `diction.pfl` file that ships with the toolkit while the other could contain all of the enhancements needed for a specific application of the toolkit. This utility could also be used multiple times to combine many different change files into a single file that would then be merged with the existing `diction.pfl` file.

It is stipulated that the input files used by mc3dcomb be formatted just like `diction.pfl`. Given that, both files then need to be sorted in ascending attribute order. That is, the order in each particular file must be ascending, but there doesn't need to be any correlation in order between the two files.

The appendix should be consulted in order to see the format of `diction.pfl` files.

The usage for mc3dcomb is the following:

```
mc3dcomb <diction.pfl> <input_file> <output_file>
```

## Using mc3icomb

The mc3icomb utility functions differently than does mc3dcomb in that it must do more processing on input files. mc3icomb must read in its configuration file (which contains the changes that a user wishes to make), read in the existing mergecom.srv, and read in the existing info.pfl. It must then merge all of this data into the proper format for the two files which it will output.

As with mc3dcomb, mc3icomb can be used on multiple files, multiple times. This enables the user to use the output of one pass or run of mc3icomb as the input for another pass of mc3icomb.

mc3icomb uses the following types of files for input:

```
mergecom.srv
info.pfl
config
```

The mergecom.srv file can be the mergecom.srv file that was shipped with the toolkit. It can be any file that is formatted and contains information for a mergecom.srv file.

The info.pfl file can be the info.pfl file that was shipped with the toolkit. It can be any file that is formatted and contains information for an info.pfl file.

The config file is a file which contains information that the user wishes to add to a set of existing info.pfl and mergecom.srv files. The format of the config file is specific to this type of file and will be detailed later.

mc3icomb creates the following output files:

```
mergecom.srv
info.pfl
```

The two output files contain all of the changes specified within the config (input) file.

### Format of mc3icomb's configuration file

An example mc3icomb configuration file is shipped with the toolkit. It is also reproduced in its entirety, here. Each section's functionality will be discussed after it is enumerated:

```
#
# This is an example of the configuration file that is used
# by mc3icomb to modify a mergecom.srv and info.pfl file.
# This file contains three sections.  Each section must be
# formatted properly in order for mc3icomb to parse it
# correctly.
#
# The following are examples of how each section should be
# constructed.
```

The above block is nothing more than a comment block that describes the purpose of the file. As can be seen by this example, comments are lines that start with the "#" character.

```
# Section one contains any additions to standard (pre-
# existing) messages or items.
# This is where private attributes are usually added to
# existing services.

# This section can be blank if you aren't using it, but the # line
containing the EXISTING keyword must remain as is.

       [ EXISTING_SERVICE_SECTION ]

       [ BASIC_COLOR_IMAGE_BOX, N_SET_RSP ]
       0009,ACME Imaging Group,2A,@PN@3@@;
       0009,ACME Imaging Group,2B,@DT@3@@;
```

As is mentioned in the comment block for this section, this is section one. The start of this section is denoted by the “EXISTING_SERVICE_SECTION” keyword. This keyword must exist in the file and must be formatted exactly as it looks in the above example. It must be enclosed in square brackets, have a space between the bracket and the keyword, and be separated from the section’s actual modifications by a blank line. All keywords contained within this file must be formatted in this manner.

The actual modifications are following the keyword delimiter and the blank line. The modifications are formatted in the following manner:

```
       [ Existing service, existing command ]
       Private Group,Creator Code,Element,@VR@VT@@;
```

For more information regarding the format of the individual modification lines, please see the appendix. The appendix describes the format of these lines in the section that describes adding private attributes to a message object (SOP).

Section two begins after the section one modifications are placed in the `config` file. The file contains a comment block describing the second section of the file. The comment block is followed by another delimiter. The delimiter is “PRIVATE_SERVICE_SECTION”.

```
       # Section two contains any private services which are going
       # to be created.
       # As with section one, this section can remain empty but the
       # PRIVATE_SERVICE_SECTION keyword must also remain exactly
       # as listed below.

       [ PRIVATE_SERVICE_SECTION ]

       [ BASIC_DUMMY_MANAGEMENT, MESSAGE ]
              UID                    = 1.2.3.4.5.6
              DESCRIPTION            = Basic dummy management
              TYPE                   = BASE

       [ BASIC_DUMMY_MANAGEMENT, N_SET_RQ ]
       2030,0010@US@1@@;
       2030,0020@LO@3@@;
       [ BASIC_DUMMY_MANAGEMENT, N_SET_RSP ]
       2030,0010@US@1@@;
       2030,0020@LO@1@@;
```

Each newly added service is formatted like the example above. Each service is delimited from the others by having its name contained within a delimiter. In the example above, this name is “BASIC_DUMMY_MANAGEMENT”. As you can see this service name corresponds to the definitions for each of this service’s commands, below.

The service itself must have its UID, Description, and Type defined. This is defined in the heading section.

Following the service's definition, the message's command and the tags that make up the command are entered. It is important to note that the tags that each command contains can be either private or standard. It is important to note that the lines that specify these are also defined in the appendix. The example above only shows standard attributes.

Once the new private services and commands are created, the configuration file includes a description for section three. There is also a delimiter that denotes the start of section three: "PRIVATE_ITEM_SECTION". As with other delimiters, it must be followed by a blank line and properly formatted.

```
# Section three contains any private items which are to be
# created.
# This section holds the complete definition of any private
# items.
# As with the other sections, this can remain empty but the
# PRIVATE_ITM keyword line must remain exactly as listed
# below.
[ PRIVATE_ITEM_SECTION ]
```

As shown in the comment above, section three contains the additions for private items. Each private item starts with the definition of the private item's name, and is followed by the item's tag contents.

```
 [ DUMMY_ITEM, ITEM ]
0008,1150@UI@1@@;
0008,1155@UI@1@@;
```

As with other sections, section four starts with a delimiter: "EXISTING_ITEM_SECTION". This delimiter denotes the start of additions to existing items.

```
# Section four contains modifications to any existing
# standard items
# which are to be added to.  As with the other sections,
# this one can also
# remain empty but the STANDARD ITM keyword must remain
# exactly as listed
# below.
[ EXISTING_ITEM_SECTION ]
```

Section four definitions match the format of section three. In this case, the item's name must already exist in the input database. If the item does not exist, an error is logged.

```
 [ PREFORMATTED_COLOR_IMAGE, ITEM ]
0008,2222@UI@1@@;
0008,3333@UI@1@@;
```

Note that the tags listed in this definition are added to an existing item's definition. These can also be either private or existing.

Once both `mc3icomb` and `mc3dcomb` have been used to generate new configuration files, the user can continue with the generation of the binary database files. This is discussed in the next section, and is no different had the user modified the configuration files by hand.

The usage for mc3icomb is as follows:

```
mc3icomb <mergecom.srv> <info.pfl> <input_file> <output.srv>
<output.pfl>
```

### Final overview of the process

As can be seen by the above descriptions, modifying the data dictionary for the MergeCOM-3 DICOM Toolkit can be accomplished in the following manner:

1. Generate the configuration files for use with mc3icomb and mc3dcomb. These files contain the actual additions/changes to the database.
2. Generate the final set of database files that you wish to use with the toolkit. This involves using mc3icomb and mc3dcomb with your configuration files and the original ASCII database files shipped with the toolkit.
3. Use mc3dict and mc3info to generate a new set of binary dictionary files used by the toolkit.

## Generating the Binary Database

Once the dictionary and message info profiles are in the correct format, they can be translated to a binary format using the mc3dict and mc3info executables, respectively.

### Dictionary File Generation (mc3dict)

Creating the binary dictionary file is a matter of running the mc3dict application specifying a single parameter which is the dictionary profile. In the usual case where the dictionary profile is named diction.pfl the output of mc3dict looks like the following:

```
mc3dict - Generate MergeCOM-3 Binary Dictionary File
Version 2
Copyright (c) 1995 Merge Technologies Inc.

Number of Data Elements processed:
       100...200...300...400...500...600...700...800...
       804...
DONE.
```

This output shows the normal processing of a dictionary profile containing 804 entries. mc3dict also has limited error detection. If it finds an input line with an invalid format, it will alert the operator as follows:

```
mc3dict - Generate MergeCOM-3 Binary Dictionary File
Version 2
Copyright (c) 1995 Merge Technologies Inc.

Number of Data Elements processed:
       ERROR: Invalid Input File format in line number 17.
```

The output file created is always the binary dictionary file mrgcom3.dct. For the binary dictionary to be accessed properly at runtime, the mergecom.pro configuration file for the toolkit must point to the location of the DICTIONARY_FILE in the [MESSAGE_PARMS] section. See the discussion of toolkit configuration in the User's Manual for further details.

## Message Info File Generation (mc3info)

Creating the binary message info files involves running the `mc3info` application specifying a single parameter which is the message info profile. In the usual case where the message info profile is named `info.pfl` the output of `mc3info` looks like the following:

```
mc3info - Generate MergeCOM-3 Binary Message Object Files
Version 2
Copyright (c) 1995 Merge Technologies Inc.

Generating info files for:
      Service #0001, Command = N_SET_RQ...
      Service #0001, Command = N_SET_RSP...
      Service #0002, Command = N_SET_RQ...
      Service #0002, Command = N_SET_RSP...
      Service #0003, Command = N_CREATE_RQ...
      Service #0003, Command = N_CREATE_RSP...
      Service #0003, Command = N_SET_RQ...
      Service #0003, Command = N_SET_RSP...
      Service #0003, Command = N_DELETE_RQ...
      Service #0003, Command = N_DELETE_RSP...
      Service #0003, Command = N_ACTION_RQ...
      Service #0003, Command = N_ACTION_RSP...
      Service #0004, Command = N_CREATE_RQ...
      Service #0004, Command = N_CREATE_RSP...
      Service #0004, Command = N_SET_RQ...
      Service #0004, Command = N_SET_RSP...
      Service #0004, Command = N_DELETE_RQ...
      Service #0004, Command = N_DELETE_RSP...
      Service #0004, Command = N_ACTION_RQ...
      Service #0004, Command = N_ACTION_RSP...
      Service #0005, Command = N_SET_RQ...
      Service #0005, Command = N_SET_RSP...
                     .
                     .
                     .
      Service #0041, Command = C_MOVE_RQ...
      Service #0041, Command = C_MOVE_RSP...
      Service #0041, Command = C_CANCEL_RQ...
      Service #0042, Command = N_CREATE_RQ...
      Service #0042, Command = N_CREATE_RSP...
      Service #0042, Command = N_SET_RQ...
      Service #0042, Command = N_SET_RSP...
      Service #0042, Command = N_DELETE_RQ...
      Service #0042, Command = N_DELETE_RSP...
      Item #0001...
      Item #0002...
      Item #0003...
      Item #0004...
      Item #0005...
                     .
                     .
                     .
      Item #0080...
      Item #0081...
      Item #0082...
      Item #0083...
      Item #0084...
      Item #0085...
      Item #0086...
      Item #0087...
      Item #0088...
      Item #0089...
DONE.
```

This output shows the normal processing of the message info profile containing some 160 messages over 42 services, and an additional 89 items. `mc3info` also has limited error detection. If it finds an input line with an invalid format, it will alert the operator as follows:

```
mc3info - Generate MergeCOM-3 Binary Message Object Files
Version 2
Copyright (c) 1995 Merge Technologies Inc.
Generating info files for:
      Service #0001, Command = N_SET_RQ...
      Service #0001, Command = N_SET_RSP...
      Service #0002, Command = N_SET_RQ...
      Service #0002, Command = N_SET_RSP...
      Service #0003, Command = N_CREATE_RQ...
      Service #0003, Command = N_CREATE_RSP...
      Service #0003, Command = N_SET_RQ...
      Service #0003, Command = N_SET_RSP...
      Service #0003, Command = N_DELETE_RQ...
      Service #0003, Command = N_DELETE_RSP...
      Service #0003, Command = N_ACTION_RQ...
      Service #0003, Command = N_ACTION_RSP...
      Service #0004, Command = N_CREATE_RQ...
      Service #0004, Command = N_CREATE_RSP...
      Service #0004, Command = N_SET_RQ...
      Service #0004, Command = N_SET_RSP...
      Service #0004, Command = N_DELETE_RQ...
      Service #0004, Command = N_DELETE_RSP...
      Service #0004, Command = N_ACTION_RQ...
      Service #0004, Command = N_ACTION_RSP...
      Service #0005, Command = N_SET_RQ...
      Service #0005, Command = N_SET_RSP...
      Service #0006, Command = N_GET_RQ...
      Service #0006, Command = N_GET_RSP...
ERROR: Invalid Input File format in line number 183.
```

The output file created is the binary message info files `mrgcom3.msg`.

For the binary info file to be accessed properly at runtime, the `mergecom.pro` configuration file for the toolkit must point to the location of the `MSG_INFO_FILE` in the `[MESSAGE_PARMS]` section. See the discussion of toolkit configuration in the User's Manual for further details.

# Accessing Private Attributes

**Benefits for the implementor!**

Once private attributes have been added to the binary dictionary file, three things occur:

- the same private attributes can be referenced in the message info file.
- private attributes in the binary dictionary file, but not part of a message object, can be added to a message object using the `MC_Add_Private_Attribute( )` call. No,`MC_Add_Private_Block( )` call is necessary for private attributes that have been added to the dictionary file. This is consistent with how `MC_Add_Standard_Attribute( )` already works.
- when messages are received over the network, or read in from media, if the private attributes contained in that message are defined in your dictionary they will be added to the message with a known VR and VM. They will also properly appear in the output of `MC_List_Message( )`.

Once private attributes have been added to a section of the message info file, three things occur:

- private attribute can be assigned values using the `MC_Set_pValue_...( )` functions. No `MC_Add_Private_Block( )` or `MC_Add_Private_Attribute( )` calls are necessary for a private attributes that have been added to binary message info file.
- when messages are received over the network, or read in from media, if the private attributes contained in that message are defined in your corresponding message info file they will be added to the message with a known VT, and Defined Terms or Enumerated Values (if specified). They will also properly appear in the output of `MC_List_Message( )`.
- when `MC_Validate_Message( )` is called, the private attributes will also be validated against the specified parameters in the corresponding message info file.

# Appendix - Dictionary Profile Specifications

This appendix describes how to modify the data dictionary manually. It is provided as a reference for users that have a desire to perform this modification themselves rather than using the included utilities to do it. Merge recommends performing all modifications through the use of the `mc3icomb` and `mc3dcomb` utilities, however.

This appendix also provides important information on the format of Merge's database files. Some of this information is important to users of the `mc3icomb` and `mc3dcomb` utilities. Since both modification utilities rely on the formats of `diction.pfl`, `info.pfl`, and `mergecom.srv` it is important to understand how these files are formatted. Also, the modification utilities rely on these formats within their respective configuration files. A user using the modification utilities must be proficient in these file formats for the utilities to be useful.

## Data File Descriptions

The dictionary profile (`diction.pfl`) contains the DICOM Standard Data Dictionary specified in Part 6 of the DICOM Standard and is formatted, one entry per line, as follows:

```
Attribute Name,Group#,Element#,VR,VM
```

where:

- `Attribute Name` is the name of the DICOM Attribute and consists of Alphanumeric characters and the space character.
- `Group#` is the 4 high order hexadecimal digits of the Attribute Tag.
- `Element#` is the 4 low order hexadecimal digits of the Attribute Tag.
- `VR` is the two uppercase character representation of the Value Representation. It must have one of the values from the set: `{ AE, AS, CS, DA, DS, DT, IS, LO, LT, PN, SH, ST, TM, UT, UI, SS, US, AT, SL, UL, FL, FD, UNKNOWN_VR, OB, OW, OL, OF, SQ }.`
- `VM` is the Value Multiplicity string for the attribute and must have a value from the set { "1", "1-N", "N", "*k*", or "1-*k*"} where *k* is a positive integer greater than 1.
- All entries are in increasing hexadecimal order of Tag (Group#,Element#), and should NOT contain spaces around the commas ( , ) in the string.

**Example from dictionary profile**

An excerpt from Merge's standard DICOM dictionary profile is as follows:

```
.
.
.
Transducer Orientation,0008,2204,CS,1
Anatomic Structure,0008,2208,CS,1
Anatomic Region Sequence,0008,2218,SQ,1
Anatomic Region Modifier Sequence,0008,2220,SQ,1
Primary Anatomic Structure Sequence,0008,2228,SQ,1
Primary Anatomic Structure Modifier Sequence,0008,2230,SQ,1
```

```
Group 0008 Comments RETIRED,0008,4000,LO,1-N
Group 0010 Length,0010,0000,UL,1
Patient's Name,0010,0010,PN,1
Patient ID,0010,0020,LO,1
Issuer of Patient ID,0010,0021,LO,1
Patient's Birth Date,0010,0030,DA,1
Patient's Birth Time,0010,0032,TM,1
Patient's Sex,0010,0040,CS,1
.
.
.
```

## Private Attribute Specification

Private attributes can be added to the dictionary profile by adding new entries, formatted one entry per line, as follows:

```
Attribute Name,Group #,Private Code,Element Byte #,VR,VM
```

where:

- `Attribute Name` is the name of the Private Attribute and consists of Alphanumeric characters and the space character.
- `Group#` must be odd and is the 4 high order hexadecimal digits of the private Attribute Tag.
- `Private Code` is the Private Creator code that will be used to identify the block of 0xFF (255) private attributes you are reserving within private `Group #`. Private Code can be composed of up to 64 alphanumeric characters and the space character. For Private Codes of any length, a trailing space should NOT be added.
- `Element Byte #` is the 2 hexadecimal digits identifying the private attribute within the Private Block identified by `Private Code`.
- `VR`, and `VM` are defined as in the case of Standard Attributes above.
- All private tags within a single Private Block (identified by a single `Private Code`) must be grouped together.
- All entries within a Private Block are in increasing hexadecimal order of Tag (`Group #,Element Byte #`).
- There should NOT be any spaces placed around the commas ( , ) in the string.

An excerpt from Merge's standard DICOM dictionary profile with additional private attributes follows (private attributes are in boldface):

```
Transducer Orientation,0008,2204,CS,1
Anatomic Structure,0008,2208,CS,1
Anatomic Region Sequence,0008,2218,SQ,1
Anatomic Region Modifier Sequence,0008,2220,SQ,1
Primary Anatomic Structure Sequence,0008,2228,SQ,1
Primary Anatomic Structure Modifier Sequence,0008,2230,SQ,1
Group 0008 Comments RETIRED,0008,4000,LO,1-N
Last Serviced By,0009,ACME Imaging Group,2A,PN,1
Last Serviced Date,0009,ACME Imaging Group,2B,DT,1
Bold Density Scan Parms,0009,ACME MR Group,00,SS,3
Group 0010 Length,0010,0000,UL,1
Patient's Name,0010,0010,PN,1
Patient ID,0010,0020,LO,1
```

```
Issuer of Patient ID,0010,0021,LO,1
Patient's Birth Date,0010,0030,DA,1
Patient's Birth Time,0010,0032,TM,1
Patient's Sex,0010,0040,CS,1
...Message Info Profile Specification
```

The message info profile (`info.pfl`) contains the SOP or Message Object specifications for all the DICOM service-command pairs supported by the MergeCOM-3 DICOM Toolkit on the `MC_Open_Message( )` and `MC_Set_Service_Command( )` calls. This profile also contains specifications for each of the Items used in attributes of Value Representation SQ (Sequence of Items).

The message info profile is broken into sections, each section beginning with a single line with the format:

`[ ServiceNum, DIMSECommandName ]` for SOPs

or

`[ ItemNum, ITEM ]` for Items

where:

- `ServiceNum` is a four digit decimal service number, corresponding to a service number defined in the `mergecom.srv` configuration file for a particular DICOM service. Leading zeroes are required where necessary.
- `DIMSECommandName` is one set of DIMSE commands { C_STORE_RQ, C_STORE_RSP, C_GET_RQ, C_GET_RSP, C_FIND_RQ, C_FIND_RSP, C_MOVE_RQ, C_MOVE_RSP, C_ECHO_RQ, C_ECHO_RSP, N_EVENT_REPORT_RQ, N_EVENT_REPORT_RSP, N_GET_RQ, N_GET_RSP, N_SET_RQ, N_SET_RSP, N_ACTION_, N_ACTION_RSP, N_CREATE_RQ, N_CREATE_RSP, N_DELETE_RQ, N_DELETE_RSP, C_CANCEL_RQ }.
- `ItemNum` is a four digit decimal item number, corresponding to an item number defined in the `mergecom.srv` configuration file. Leading zeroes are required where necessary.
- `ITEM` is the explicit string '`ITEM`'.

## Standard Attribute Specification

Within each section are specified the standard DICOM attributes that make up the Message or Item Object. Each attribute is specified on a separate line as follows:

`Group#,Element#@VR@VT@@DefOrEnumValues;`
where:

- `Group#` is the 4 high order hexadecimal digits of the Attribute Tag.
- `Element#` is the 4 low order hexadecimal digits of the Attribute Tag.
- VR is the two uppercase character representation of the Value Representation (VR). It must have one of the values from the set { `AE, AS, CS, DA, DS, DT, IS, LO, LT, PN, SH, ST, TM, UT,`

`UI, SS, US, AT, SL, UL, FL, FD, UNKNOWN_VR, OB, OW, OL, SQ }` and shall agree with the VR specified for this same Tag in the dictionary profile. **Note**: The VR appears in both the dictionary and message info profile for performance reasons.

- VT is the value type for the attribute and must have a value from the set of values {1, 1C(*ConditionString*), 2, 2C(*ConditionString*), 3} where *ConditionString* is a string representing a condition specified in the DICOM standard for a conditional attribute. *ConditionString* shall only consist of uppercase characters and the underbar character. **Note**: The Toolkit only validates against conditions that it can validate itself. It is the DICOM application's responsibility to validate all other conditions.

- `DefOrEnumValues` for single valued elements (VM of 1) is a string containing defined terms or enumerated values of the form:

    - `D`<value1,value2,... , value*n*> or `E`<value1, value2,... , value*n*>

- Where `D` specifies defined terms, `E` specifies enumerated values, and value*k* where *k* is between 1 and n specifies a term or value, respectively. For multiple valued elements (VM greater than 1) having a value multiplicity of n,
- each value*k* can itself have one of many values. In this case `DefOrEnumValues` has the following different form:

    `D`< ($value1_1$, $value1_2$,... ,$value1_j$) ($value2_1$, $value2_2$,... ,$value2_k$) ... ($valuen_1$, $valuen_2$,... ,$valuen_l$)

    or

    `E`< ($value1_1$, $value1_2$,... ,$value1_j$) ($value2_1$, $value2_2$,... ,$value2_k$) ... ($valuen_1$, $valuen_2$,... ,$valuen_l$)

- where `D` specifies defined terms, `E` specifies enumerated values, and each parenthesized list specifies the acceptable terms or values for the *k*'th value of that multiple valued element, respectively.

- All entries within a section are in increasing hexadecimal order of Tag (Group#,Element#).

- The "at" signs (@) and commas (,) present in the string are explicit separators around which no spaces should be placed.

## Private Attribute Specification

Private Attributes can also be added to a Message Object (SOP) by adding them to the correct section. Private attributes referenced in the message profile must have also been properly defined in the dictionary profile, as described earlier. Each private attribute is specified on a separate line as follows.

```
Group #,Private Code,Element Byte #@VR@VT@@DefOrEnumValues;
```

where:

- `Group#` must be odd and is the 4 high order hexadecimal digits of the private Attribute Tag.
- `Private Code` is the Private Creator code that will be used to identify the block of 0xFF (255) private attributes you are reserving within private `Group #`. Private Code can be composed of up to 64 alphanumeric characters and the space character.
- `Element Byte #` is the 2 hexadecimal digits identifying the private attribute within the Private Block identified by `Private Code`.
- VR, VT, and `DefOrEnumValues` are defined as they were in the case of Standard Attributes above.
- All private tags within a single Private Block (identified by a single `Private Code`) within a section must be grouped together.
- All entries within a Private Block are in increasing hexadecimal order of Tag (`Group #,Element Byte #`).
- The "at signs (@) and commas (,) present in the string are explicit separators around which no spaces should be placed.

## Example message info profile entry

Looking at the `BASIC_COLOR_IMAGE_BOX` service as an example, we see the following entry for this service in the `mergecom.srv` configuration file:

```
[0002]
        NAME                    = BASIC_COLOR_IMAGE_BOX
        UID                     = 1.2.840.10008.5.1.1.4.1
        TYPE                    = BASE
        COMMAND_1               = N_SET_RQ
        COMMAND_2               = N_SET_RSP
```

**Example from message info profile**

This service has `ServiceNum` 0002 and supports the `DIMSECommandName`'s `N_SET_RQ` and `N_SET_RSP`. The corresponding entries in the message profile (`info.pfl`) look like the following:

```
[ 0002, N_SET_RQ ]
2010,0060@CS@3@@D<REPLICATE, BILINEAR, CUBIC, NONE>;
2010,0080@CS@3@@;
2020,0010@US@1@@;
2020,0020@CS@3@@E<NORMAL, REVERSE>;
2020,0030@DS@3@@;
2020,0111@SQ@1@@E<PREFORMATTED_COLOR_IMAGE>;
2020,0130@SQ@3@@E<REF_OVERLAY>;
[ 0002, N_SET_RSP ]
2010,0060@CS@3@@D<REPLICATE, BILINEAR, CUBIC, NONE>;
2010,0080@CS@3@@;
2020,0010@US@1@@;
2020,0020@CS@1@@E<NORMAL, REVERSE>;
2020,0030@DS@3@@;
2020,0111@SQ@1@@E<PREFORMATTED_COLOR_IMAGE>;
2020,0130@SQ@1C(OVERLAY_SOP_NEGOTIATED)@@E<REF_OVERLAY>;
```

**Example item**

Note that both messages contain attributes with VR's of SQ. Only in these cases an enumerated value is specified for the attribute that contains the Item Name. This Item Name translates to item number in the mergecom.srv configuration file as follows:

```
[ITEM_TABLE]
    NUMBER_OF_ITEMS_SUPPORTED              = 42
    ADMITTING_DIAGNOSIS_CODE               = 0001
    DISCHARGE_DIAGNOSIS_CODE               = 0002
    INSTITUTION_CODE                       = 0003
                              .
                              .
                              .
PREFORMATTED_COLOR_IMAGE        = 0008
    .
                              .
                              .
    REF_OVERLAY                            = 0023
                              .
                              .
                              .
```

The format of the actual item in the message info profile is identical to that for the message object specified above, except for the name of the section. For example, the entry in the message info profile (`info.pfl`) for the REF_OVERLAY item is:

```
[ 0023, ITEM ]
0008,1150@UI@1@@;
0008,1155@UI@1@@;
```

This allows for nested sequences of items, since an attribute of an item can certainly have a VR of SQ.

## Extending the Message Info Profile

Extending the message or item objects specified in the message info profile with additional private attributes is a matter of adding private attribute entries to the proper section of the file. Use the mergecom.srv file to determine the proper section; namely:

- when adding private attributes to a message object, look for the service name in the mergecom.srv file, find its corresponding `ServiceNum`, and look for the section `[ServiceNum, DIMSECommandName]` where DIMSECommandName is the DIMSE command for which you wish to add the attributes.
- when adding private attributes to an item object, look for the item name in the mergecom.srv file, find its corresponding `ItemNum`, and find the section labeled [`ItemNum`, ITEM].

Once finding the proper section add a validly formatted entry for your new private attribute with all the required parameters. Be sure that the attribute's Tag has a corresponding entry in the data dictionary profile. Also, be certain that the new attribute is added in Tag order.

**Adding private attributes to the message**

A couple of the private attributes we defined earlier in the dictionary profile could be added to the `N_SET_RSP` entry above as follows (private attributes are in boldface):

```
[ 0002, N_SET_RQ ]
```

```
2010,0060@CS@3@@D<REPLICATE, BILINEAR, CUBIC, NONE>;
2010,0080@CS@3@@;
2020,0010@US@1@@;
2020,0020@CS@3@@E<NORMAL, REVERSE>;
2020,0030@DS@3@@;
2020,0111@SQ@1@@E<PREFORMATTED_COLOR_IMAGE>;
2020,0130@SQ@3@@E<REF_OVERLAY>;
[ 0002, N_SET_RSP ]
0009,ACME Imaging Group,2A@PN@3@@;
0009,ACME Imaging Group,2B@DT@3@@;
2010,0060@CS@3@@D<REPLICATE, BILINEAR, CUBIC, NONE>;
2010,0080@CS@3@@;
2020,0010@US@1@@;
2020,0020@CS@1@@E<NORMAL, REVERSE>;
2020,0030@DS@3@@;
2020,0111@SQ@1@@E<PREFORMATTED_COLOR_IMAGE>;
2020,0130@SQ@1C(OVERLAY_SOP_NEGOTIATED)@@E<REF_OVERLAY>;
```

Also, private attributes can be added to items as they were to messages.

**Important Notes**

Be careful when setting the Value Type (VT) for private attributes you are adding to the message info profile. VT for private attributes should always be 3 (meaning the private attribute is optional) unless you are defining your own Specialized or Private SOP Class.

Also, once you add private attributes to a message section in the message info file, the toolkit will reserve the private block of attributes even if your application does not fill these attributes in with values. This means the private creator codes will be a part of the message stream. If you have 'optional' blocks of private attributes you will probably want to continue to use the `MC_Add_Private_...()` runtime private attribute creation calls.